信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture
Cache IV Set-Associative Cache

Instructors: Siting Liu & Yuan Xiao
Course website: https://faculty.sist.shanghaitech.edu.cn/liust/courses/CS110.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2026/5/7

Administratives

- Mid-term II tentatively May 21st 8am-10am; you can bring 2-page A4-sized double-sided cheat sheet, **handwritten** only! No printing! (Location: TBD);
- Project 2.1 released, start early, ddl approaching!!!
- Project 2.2 to be released soon. Keep an eye on Piazza/webpage post.
- HW5 to be released, on-line questions on cache, ddl May. 14th.
- Lab 9 checking this week. Lab 10 will be released and will be checked next week (longan nano RISC-V board will also be distributed next week).
  - CS110-only students need the board for lab 10 & 11 only;
  - CS110P project 4 will use this board to develop a game;
  - Keep them really well, because you have to return the board after lab/project checking (get 0 mark if device not returned);
- Discussion this week on pipeline/superscalar;
- Discussion May. 15th on midterm II review.

Cache Design: Placement Policies

Fully Associative Cache

Put a new line anywhere

Set-Associative Cache

(This lecture)

Put a new line in several specific places

Direct Mapped Cache

Put a new line in one specific place

Fully associative caches need expensive hardware.

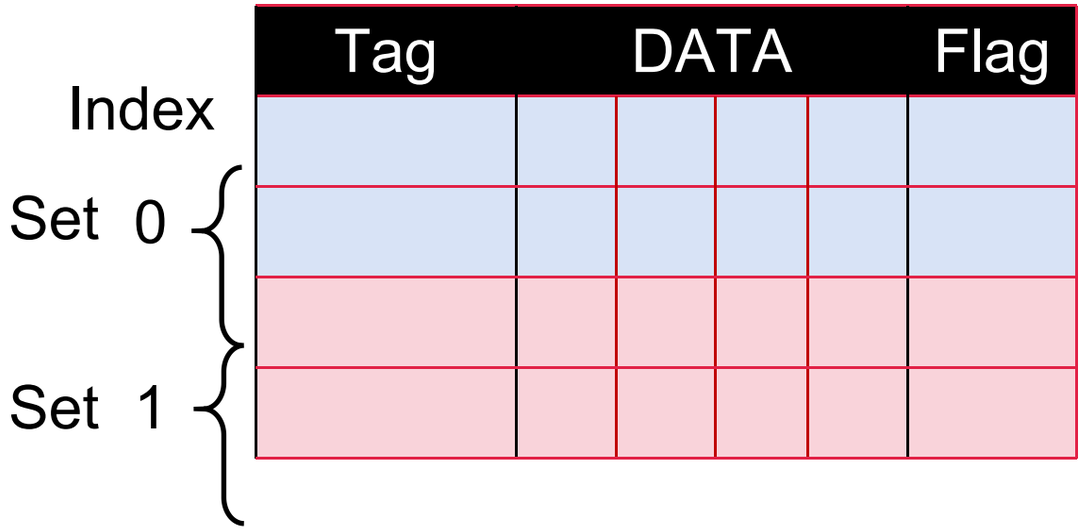
Less hardware.

Memory blocks >> # Cache blocks

We need to carefully place memory blocks into cache blocks

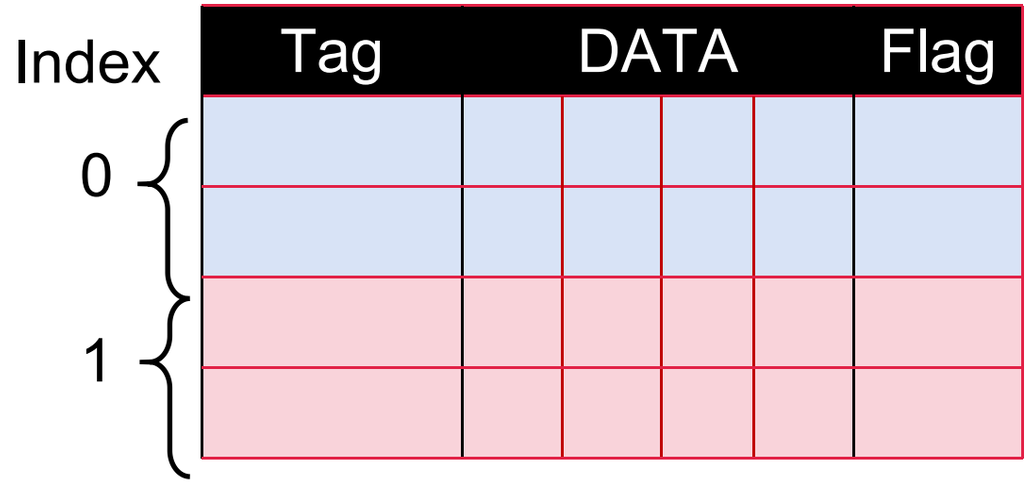
Set-Associative (SA) Cache

- Placement policy: The data can only be stored at one index, but there are multiple slots/blocks.
  - To check for existence in the cache, we check all cache blocks associated with the same index.
- Associativity of a cache: The number of slots assigned to each index/set.



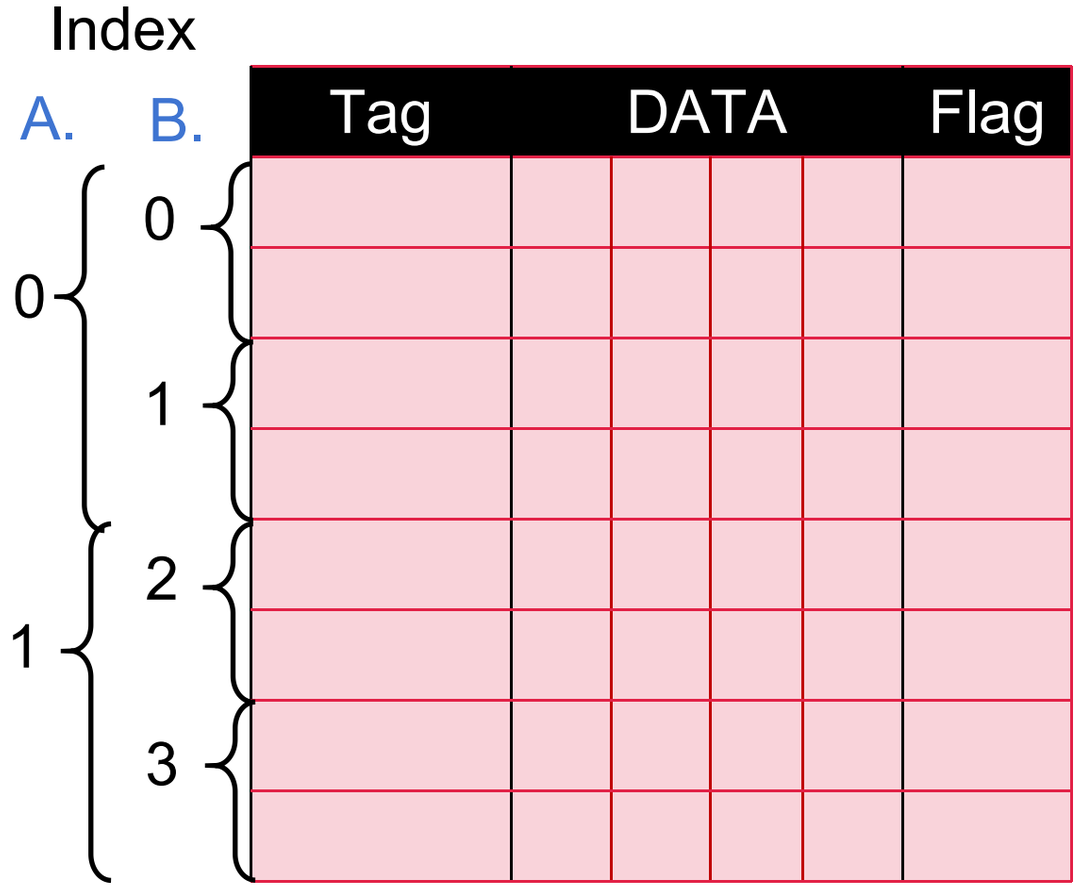
N-Way Set-Associative Cache

- Associativity of a cache: The number of slots assigned to each index.
  - N-Way Set Associative: groups of N cache lines/blocks/slots are assigned a unique index
- 2-Way Set Associative (see right)
  - 4 cache blocks
  - Each index is associated with a set of N = 2 lines
  - Two indices: 0 and 1



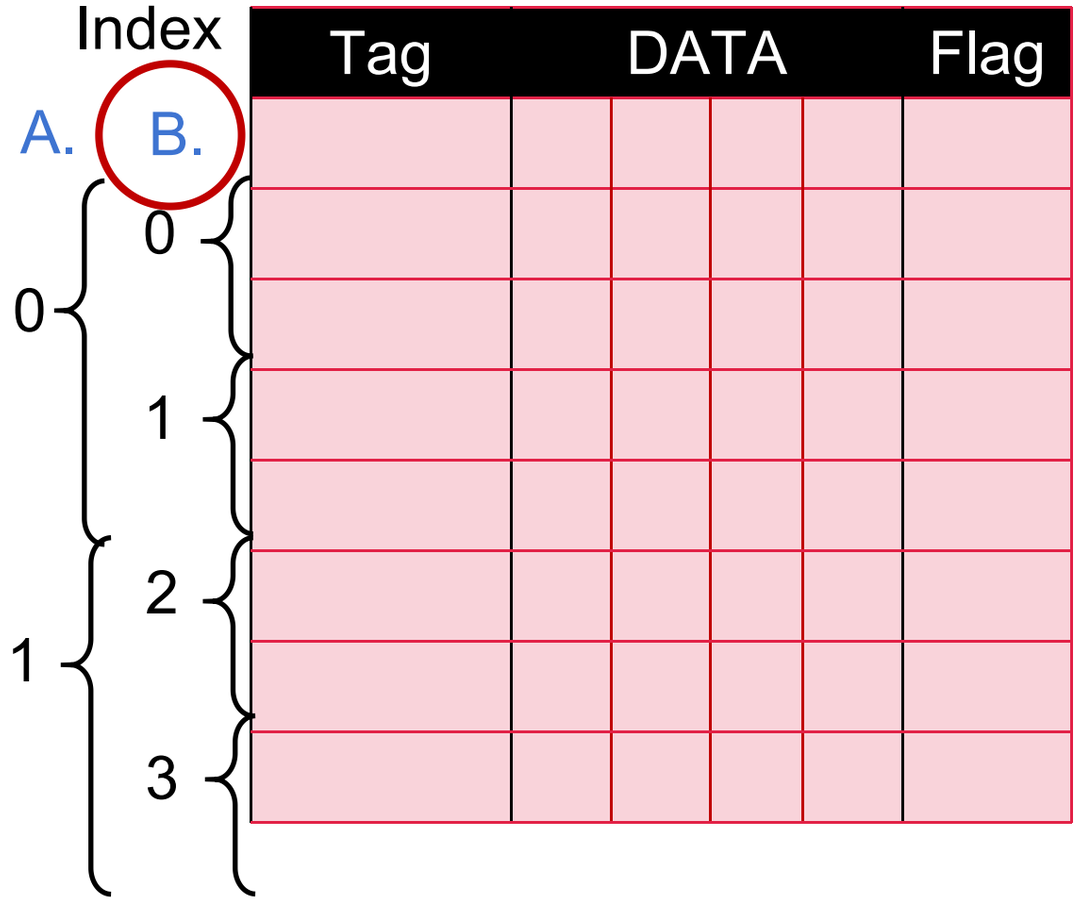
Designing N-Way SA Cache

- N-Way Set Associative: groups of N cache blocks are assigned a unique index.
  1. Which mapping represents a 32B 2-way SA cache with 4B cache blocks?
  2. Do SA caches have tags? Indices? Offsets?
  3. Do SA cache blocks need a valid bit?
  4. What write policies can SA caches support?
  5. SA caches support block replacement policies. How does LRU work?



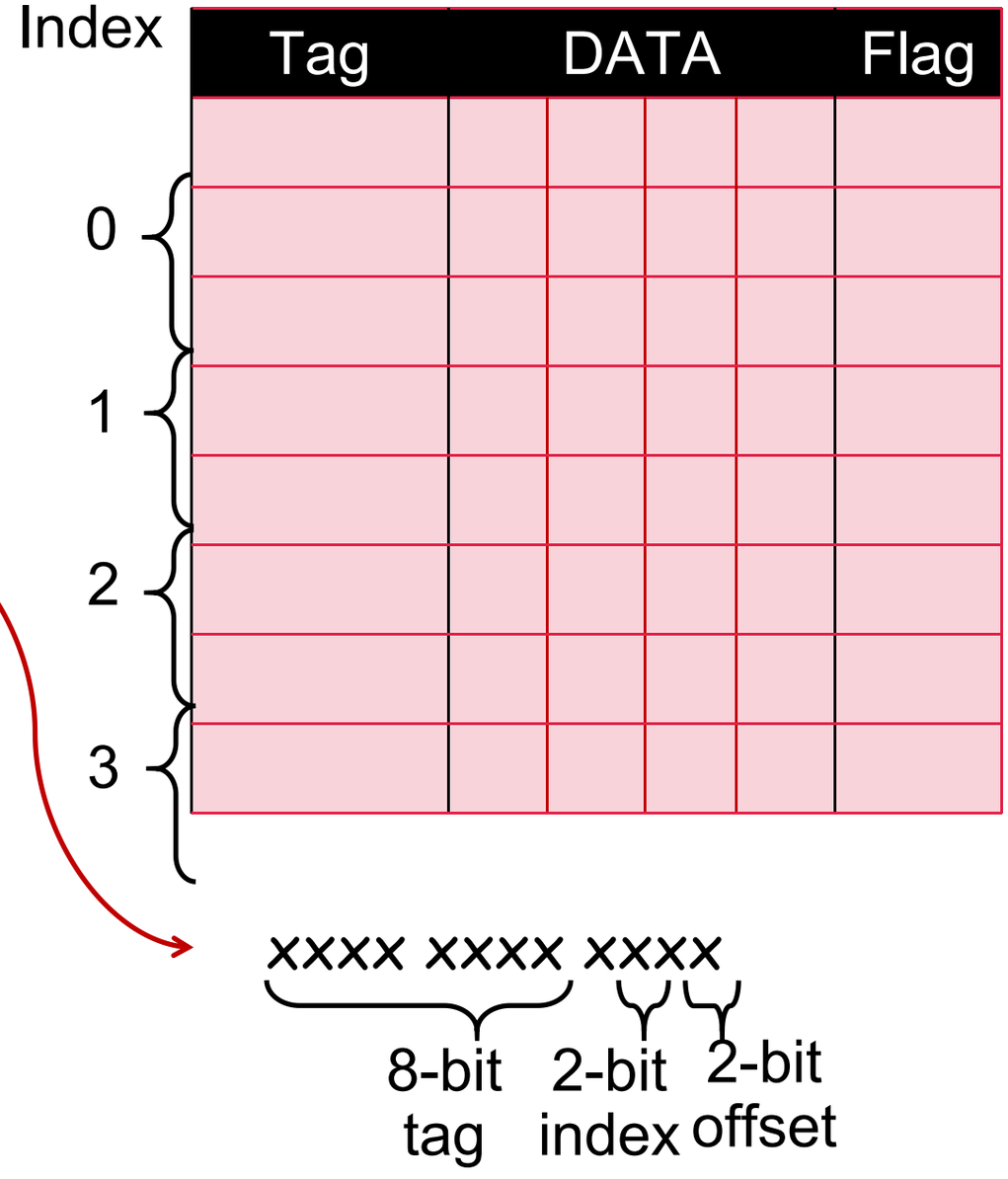
Designing N-Way SA Cache

- N-Way Set Associative: groups of N cache blocks are assigned a unique index.
  1. Which mapping represents a 32B 2-way SA Cache with 4B cache blocks?
  2. Do SA caches have Tags? Indices? Offsets?
  3. Do SA cache blocks need a valid bit?
  4. What write policies can SA Caches support?
  5. SA Caches support block replacement policies. How does LRU work?



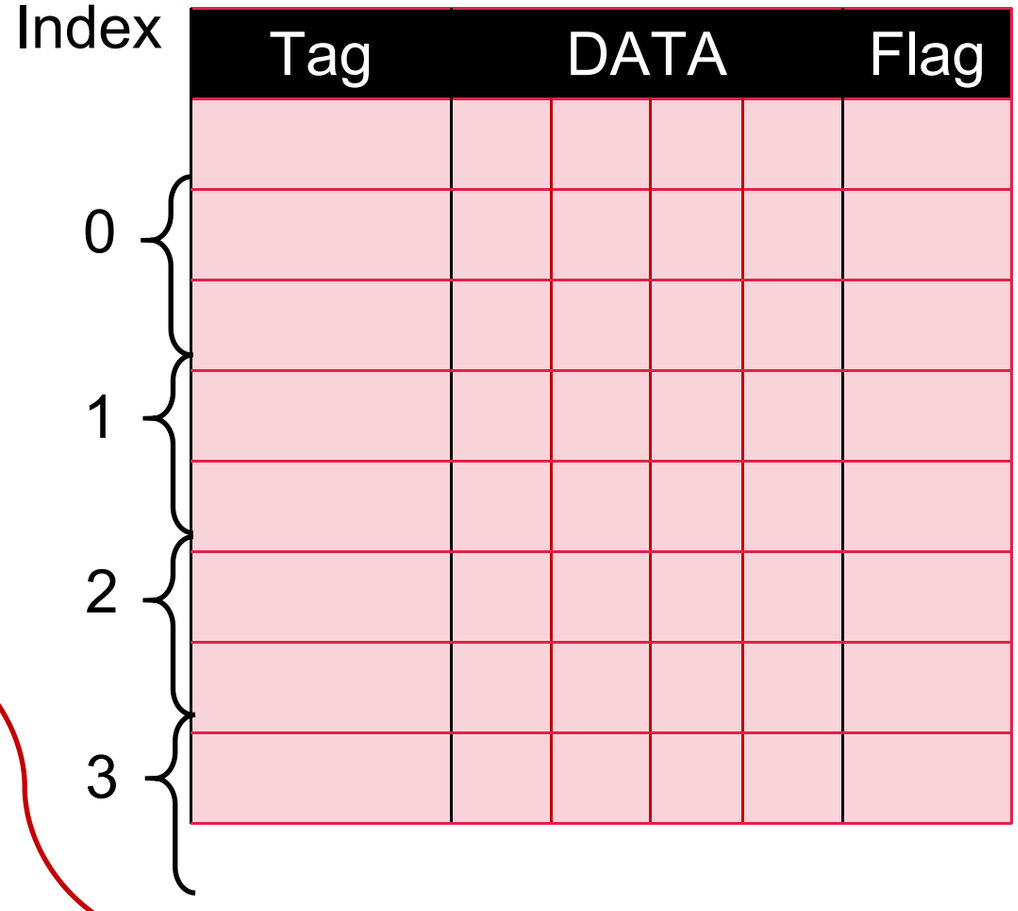
Designing N-Way SA Cache

- N-Way Set Associative: groups of N cache blocks are assigned a unique index.
  1. Which mapping represents a 32B 2-way SA cache with 4B cache blocks?
  2. Do SA caches have tags? Indices? Offsets? (assume 12-bit address)
  3. Do SA cache blocks need a valid bit?
  4. What write policies can SA caches support?
  5. SA caches support block replacement policies. How does LRU work?



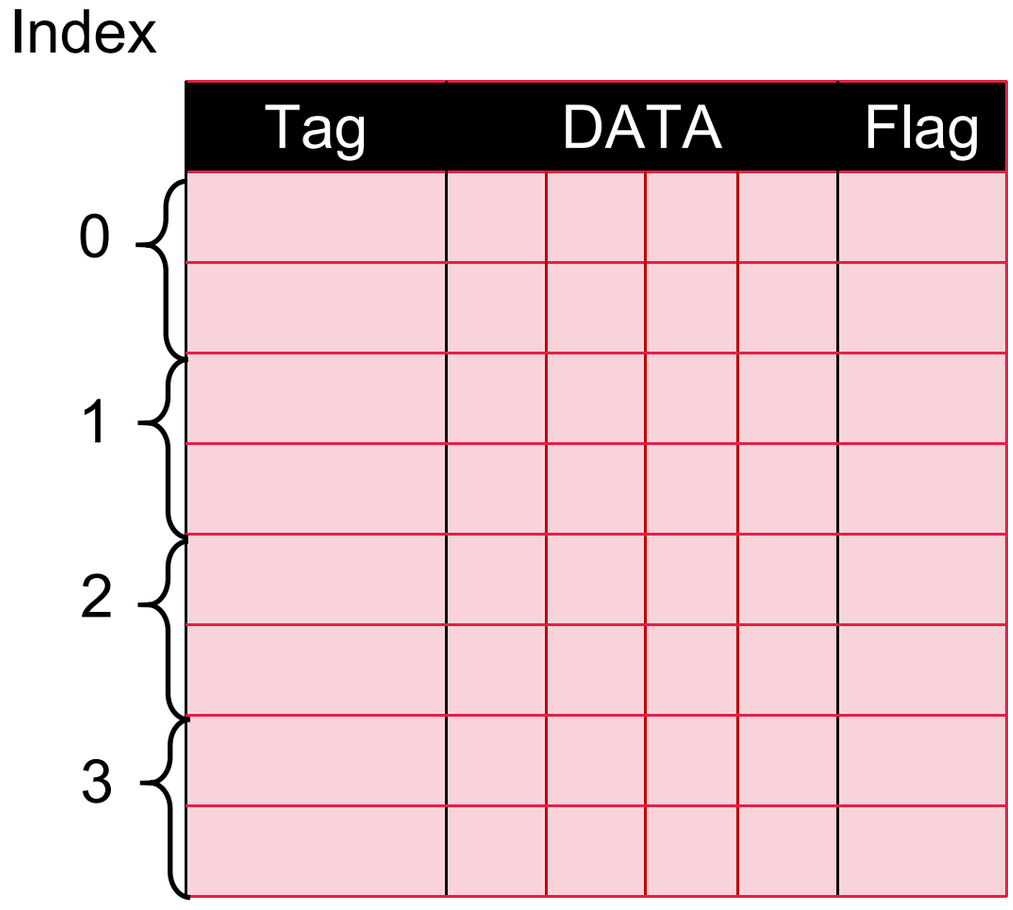
Designing N-Way SA Cache

- N-Way Set Associative: groups of N cache blocks are assigned a unique index.
  1. Which mapping represents a 32B 2-way SA Cache with 4B cache blocks?
  2. Do SA caches have tags? Indices? Offsets?
  3. Do SA cache blocks need a valid bit?
  4. What write policies can SA caches support?
  5. SA caches support block replacement policies. How does LRU work?



Designing N-Way SA Cache

- N-Way Set Associative: groups of N cache blocks are assigned a unique index.
  1. Which mapping represents a 32B 2-way SA Cache with 4B cache blocks?
  2. Do SA caches have tags? Indices? Offsets?
  3. Do SA cache blocks need a valid bit?
  4. What write policies can SA Caches support?
  5. SA Caches support block replacement policies. How does LRU work?



Write-back (needs dirty bit), write-through

Designing N-Way SA Cache

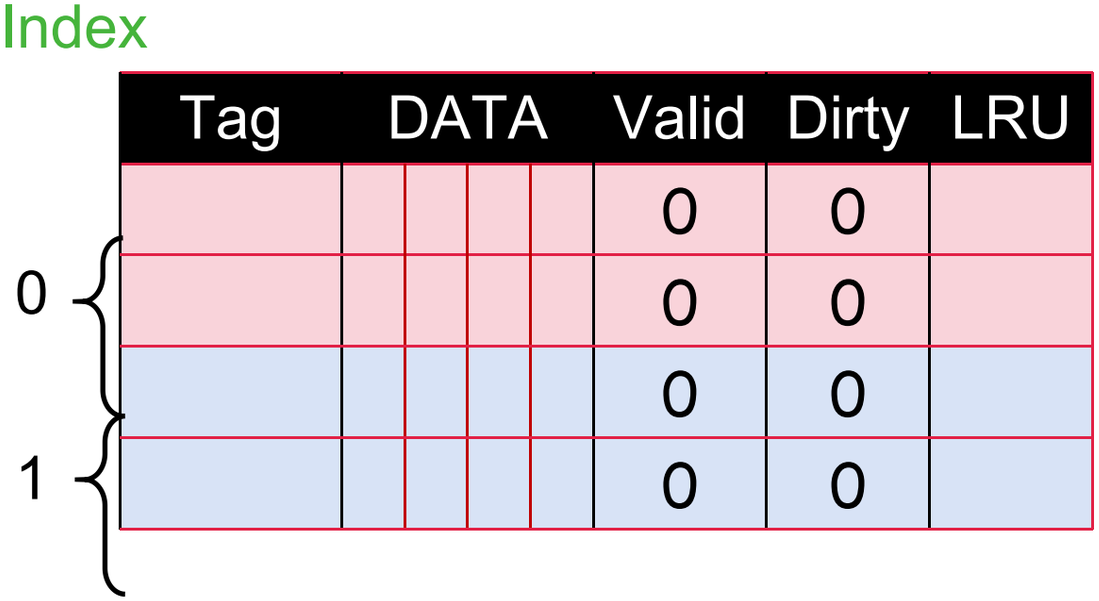
- N-Way Set Associative: groups of N cache blocks are assigned a unique index.
  1. Which mapping represents a 32B 2-way SA Cache with 4B cache blocks?
  2. Do SA caches have Tags? Indices? Offsets?
  3. Do SA cache blocks need a valid bit?
  4. What write policies can SA Caches support?
  5. SA Caches support block replacement policies. How does LRU work?



LRU (and all block replacement policies) occur within each set of cache blocks. Above, LRU considers N = 2 blocks.

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
  - 0100 0011 1011



N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
  - 0100 0011 1011

Index

	Tag	DATA				Valid	Dirty	LRU
0	0x087	d	a	t	a	1	0	0
						0	0	
1						0	0	
						0	0	

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
  - 0100 0011 0000

Index

	Tag	DATA				Valid	Dirty	LRU
0	0x087	d	a	t	a	1	0	1
	0x086	d	a	t	a	1	0	0
1						0	0	
						0	0	

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
  - 0100 0011 1000

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	1
0	0x086	data	1	0	0
			0	0	
1			0	0	

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
  - 0100 0011 1000

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
	0x086	data	1	0	1
1			0	0	
			0	0	

Cache HIT!!!

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
  - 0000 0011 0100

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
0	0x086	data	1	0	1
			0	0	
1			0	0	

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
  - 0000 0011 0100

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
0	0x086	data	1	0	1
0	0x006	data	1	0	0
1			0	0	

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
- 5. Load byte @0x836
  - 1000 0011 0110

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
	0x086	data	1	0	1
	0x006	data	1	0	0
1			0	0	

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
- 5. Load byte @0x836
  - 1000 0011 0110

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
	0x086	data	1	0	1
	0x006	data	1	0	1
1	0x106	data	1	0	0

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
- 5. Load byte @0x836
- 6. Load byte @0x234
  - 0010 0011 0100

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
	0x086	data	1	0	1
	0x006	data	1	0	1
1	0x106	data	1	0	0

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
- 5. Load byte @0x836
- 6. Load byte @0x234
  - 0010 0011 0100

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
	0x086	data	1	0	1
1	0x006	data	1	0	1
	0x106	data	1	0	0

Victim

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
- 5. Load byte @0x836
- 6. Load byte @0x234
  - 0010 0011 0100

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
	0x086	data	1	0	1
	0x046	data	1	0	0
1	0x106	data	1	0	1

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
- 5. Load byte @0x836
- 6. Load byte @0x234
- 7. Store byte @0x789, 0x0 (write allocate)
- 0111 1000 1001

Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	0
	0x086	data	1	0	1
	0x046	data	1	0	0
1	0x106	data	1	0	1

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit address
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
- 5. Load byte @0x836
- 6. Load byte @0x234
- 7. Store byte @0x789, 0x0 (write allocate)
- 0111 1000 1001

Index

Tag	DATA	Valid	Dirty	LRU
0x087	data	1	0	0
0x086	data	1	0	1
0x046	data	1	0	0
0x106	data	1	0	1

Victim

1

N-Way SA Cache-Example

- Assume 2-Way 16B Set-Associative, 4B block size, LRU within each set, write-back, cold start, 12-bit addr.
- 1. Load byte @0x43B
- 2. Load byte @0x430
- 3. Load byte @0x438
- 4. Load byte @0x034
- 5. Load byte @0x836
- 6. Load byte @0x234
- 7. Store byte @0x789, 0x0 (write allocate)
- 0111 1000 1001

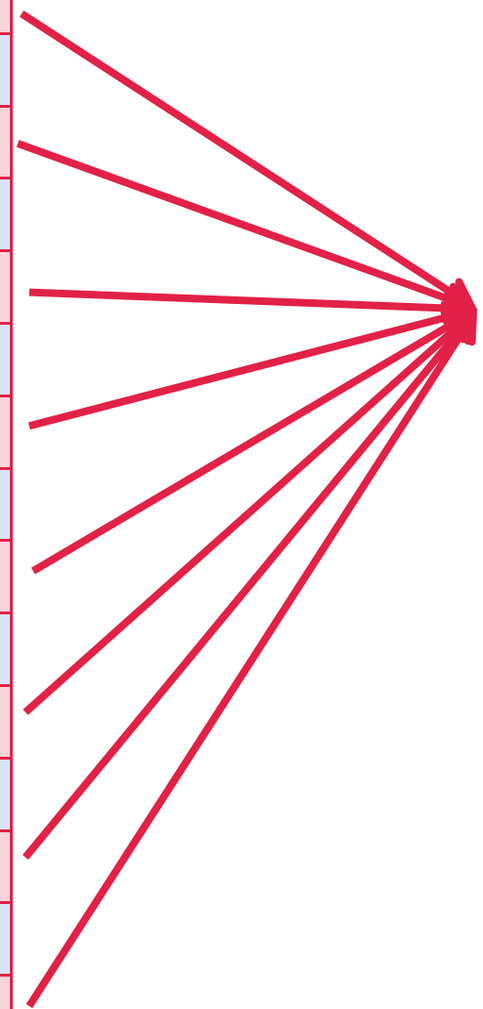
Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	1
	0x0F1	da**0**a	1	1	0
1	0x046	data	1	0	0
	0x106	data	1	0	1

N-Way SA Cache Mapping Patterns

- Assume 2-way set-associative, 4B block size; 6-bit address;

Address		
Tag	Index	Offset
000	0	xx
000	1	xx
001	0	xx
001	1	xx
010	0	xx
010	1	xx
011	0	xx
011	1	xx
100	0	xx
100	1	xx
101	0	xx
101	1	xx
110	0	xx
110	1	xx
111	0	xx
111	1	xx



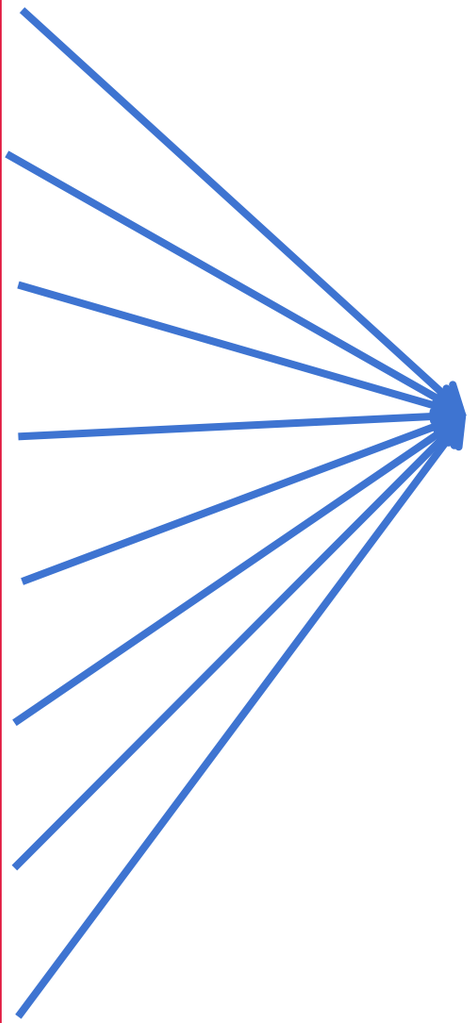
Index

Index	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	1
	0x0F1	da0a	1	1	0
1	0x046	data	1	0	0
	0x106	data	1	0	1

N-Way SA Cache Mapping Patterns

- Assume 2-way set-associative, 4B block size; 6-bit address;

Address		
Tag	Index	Offset
000	0	xx
000	1	xx
001	0	xx
001	1	xx
010	0	xx
010	1	xx
011	0	xx
011	1	xx
100	0	xx
100	1	xx
101	0	xx
101	1	xx
110	0	xx
110	1	xx
111	0	xx
111	1	xx



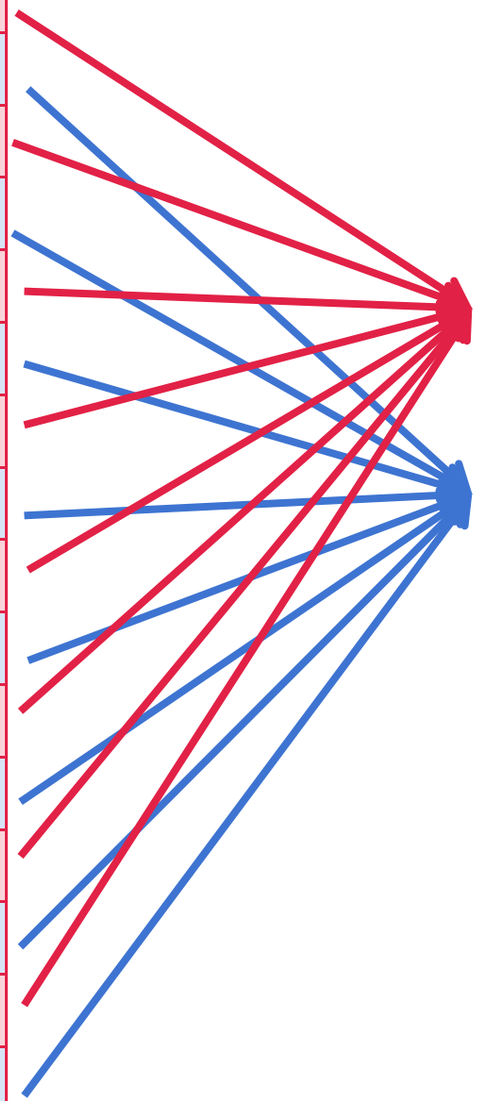
Index

	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	1
	0x0F1	da0a	1	1	0
1	0x046	data	1	0	0
	0x106	data	1	0	1

N-Way SA Cache Mapping Patterns

- Assume 2-way set-associative, 4B block size; 6-bit address;

Address		
Tag	Index	Offset
000	0	xx
000	1	xx
001	0	xx
001	1	xx
010	0	xx
010	1	xx
011	0	xx
011	1	xx
100	0	xx
100	1	xx
101	0	xx
101	1	xx
110	0	xx
110	1	xx
111	0	xx
111	1	xx



Index

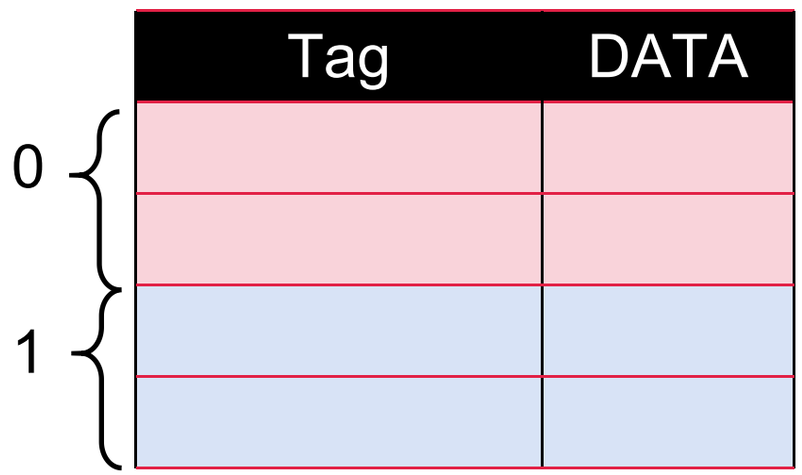
	Tag	DATA	Valid	Dirty	LRU
0	0x087	data	1	0	1
	0x0F1	data	1	1	0
1	0x046	data	1	0	0
	0x106	data	1	0	1

Conflict Miss is Mitigated

- **C**ompulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- **C**apacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- **C**onflict (**C**ollision):
  - Multiple memory locations mapped to the same cache location, conflict even when the cache has not reached full capacity.
  - Solution 1: increase cache size
  - Solution 2: increase associativity (may increase access time)

Conflict Miss is Mitigated

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index i = 1-bit width; 2-way set-associative
- t = 12 - 2 - 1 = 9 bit width;
- Load the 0th element and the 4th element of an int array a altenately;
- Cold start with an empty cache;
  - xxxx xxx0 0000 Address of the 0th element



Conflict Miss is Mitigated

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index i = 1-bit width; 2-way set-associative
- t = 12 - 2 - 1 = 9 bit width;
- Load the 0th element and the 4th element of an int array a altenately;
- Cold start with an empty cache;
  - xxxx xxx0 0000 Address of the 0th element



Conflict Miss is Mitigated

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index i = 1-bit width; 2-way set-associative
- t = 12 - 2 - 1 = 9 bit width;
- Load the 0th element and the 4th element of an int array a altenately;
- Cold start with an empty cache;
  - xxxx xxx0 0000 Address of the 0th element
  - xxxx xxx1 0000 Address of the 4th element



Conflict Miss is Mitigated

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index i = 1-bit width; 2-way set-associative
- t = 12 - 2 - 1 = 9 bit width;
- Load the 0th element and the 4th element of an int array a altenately;
- Cold start with an empty cache;
  - xxxx xxx0 0000 Address of the 0th element
  - xxxx xxx1 0000 Address of the 4th element
  - xxxx xxx0 0000 Address of the 0th element

Cache Hit

	Tag	DATA
0	xxxxxxx00	a[0]
	xxxxxxx10	a[4]
1		

Conflict Miss is Mitigated

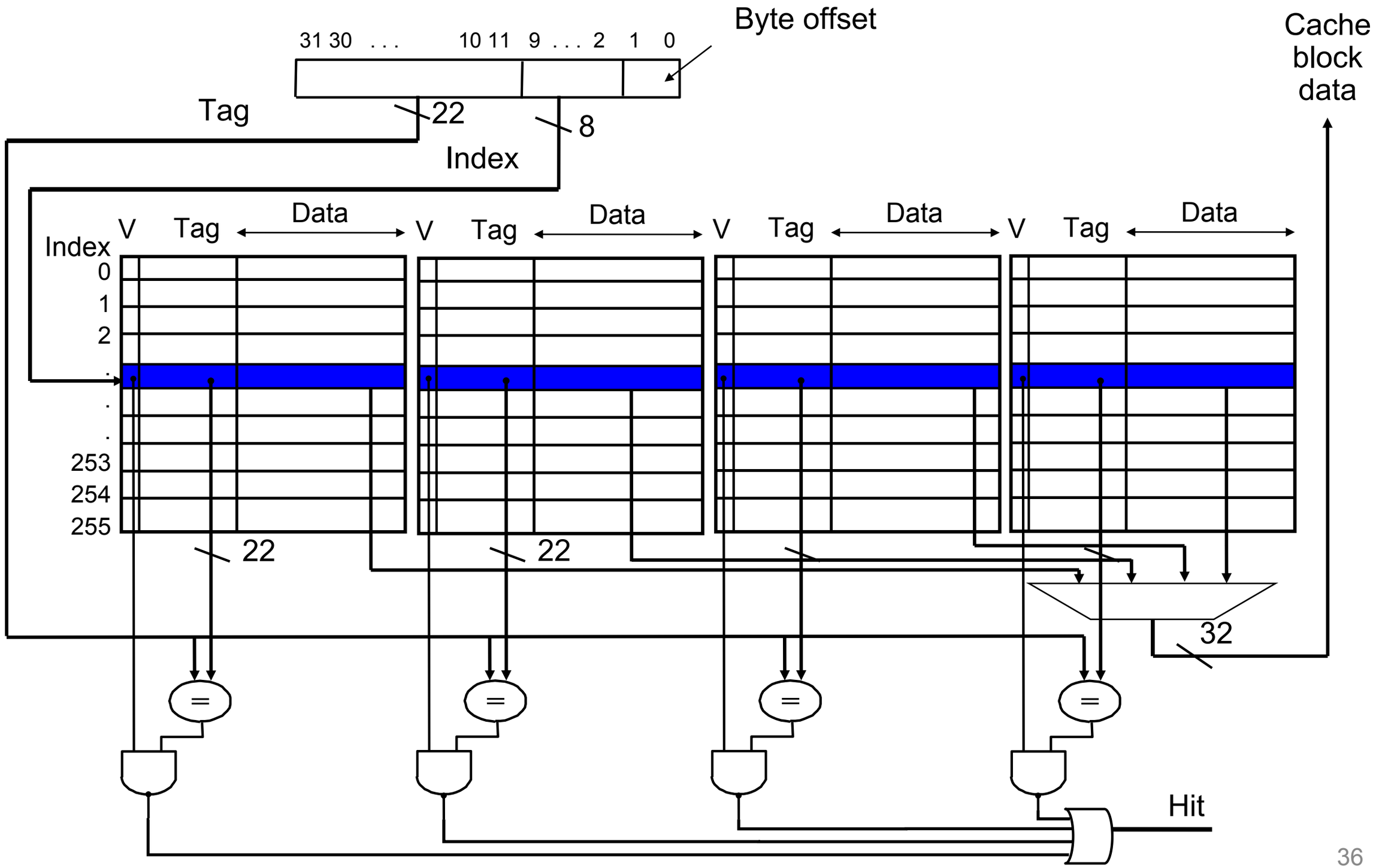
- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index i = 1-bit width; 2-way set-associative
- t = 12 - 2 - 1 = 9 bit width;
- Load the 0th element and the 4th element of an int array a altenately;
- Cold start with an empty cache;
  - xxxx xxx0 0000 Address of the 0th element
  - xxxx xxx1 0000 Address of the 4th element
  - xxxx xxx0 0000 Address of the 0th element
  - xxxx xxx1 0000 Address of the 4th element

Cache Hit

	Tag	DATA
0	xxxxxxx00	a[0]
0	xxxxxxx10	a[4]
1		
1		

Avoid the ping-pong effect

No Free Lunch-Hardware Implementation



Set-Associative Cache Terminology

- Cache capacity/size: total size of the cache (C)
- Cache block size: C_B → decides the number of offset bits (o)

$$2^o = C_B$$

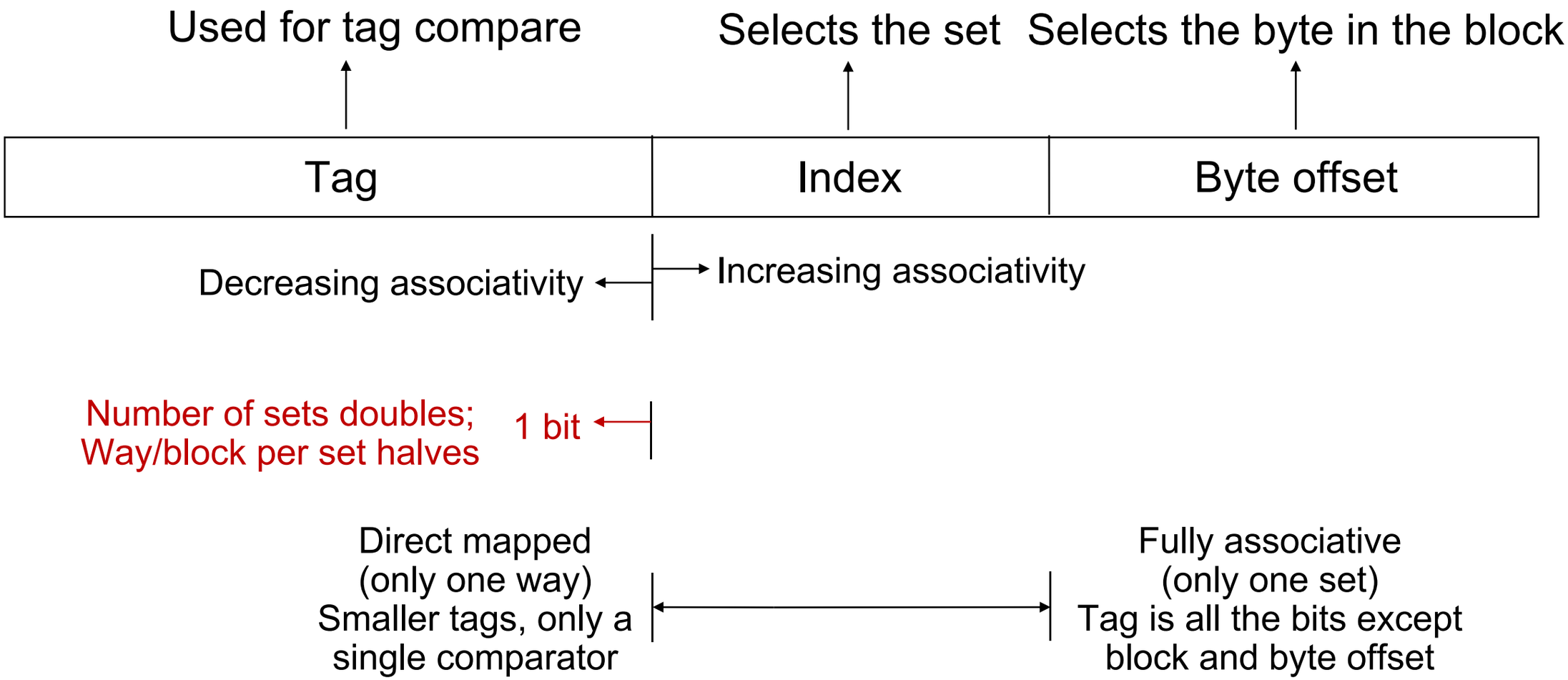
- Number of cache blocks (#cache block, M)

$$M * C_B = C$$

- Bit width of memory address (w)
- N-way SA cache: N cache blocks in a set
- #set = M/N
- Bit width of Index (i): $\log_2$(#set) or $\log_2$(#M/N)
- Bit width of Tag (t): t = w - o - i bits
- Hardware implication: comparators? actual storage requirement?
- Given N-way, t, i and o, total capacity $C = 2^o * 2^i * N$

Tune the Knobs

- Given N-way, t, i and o, total capacity = 2^o * (2^i * N)



Tune the Knobs

- Given N-way, t, i and o, total capacity = 2^o * (2^i * N)
- Example: different organizations of an 8-block cache

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

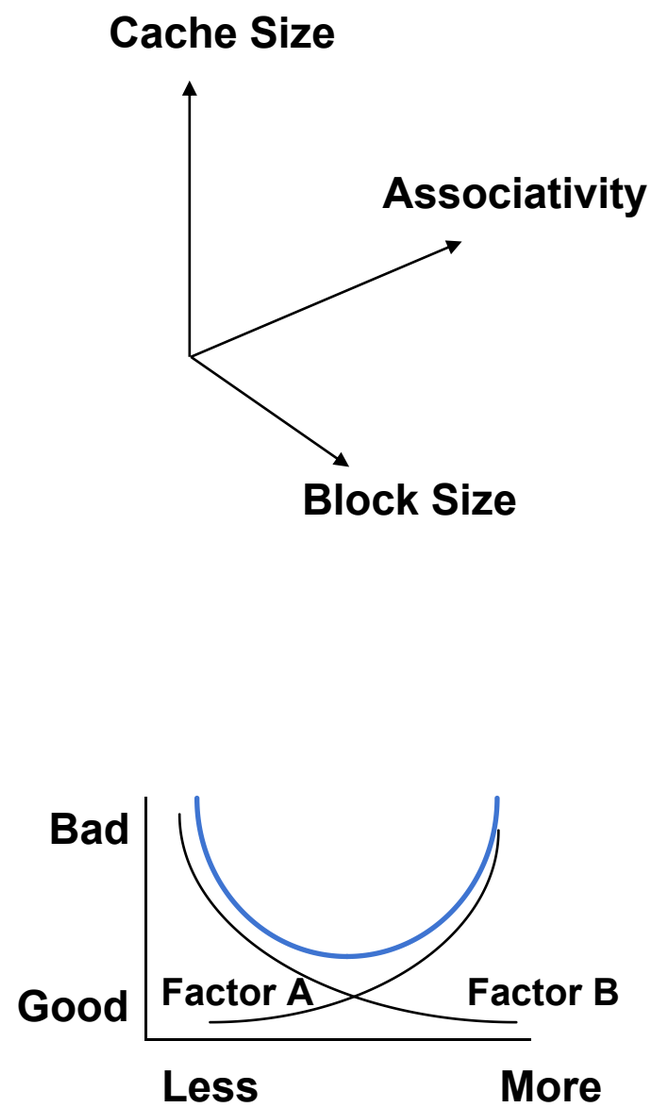
Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Total number blocks is equal to *number of sets* × *associativity*. For fixed cache size and fixed block size, increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative cache is same as a fully associative cache.

Cache Design Space

- Several interacting dimensions
  - Cache size
  - Block size
  - Associativity
  - Replacement policy
  - Write-through vs. write-back
  - Write allocation
- Optimal choice is a compromise
  - Depends on access characteristics
  - Workload
  - Use (I-cache, D-cache)
  - Depends on technology / cost
- Simplicity often wins



Cache Performance & Metrics

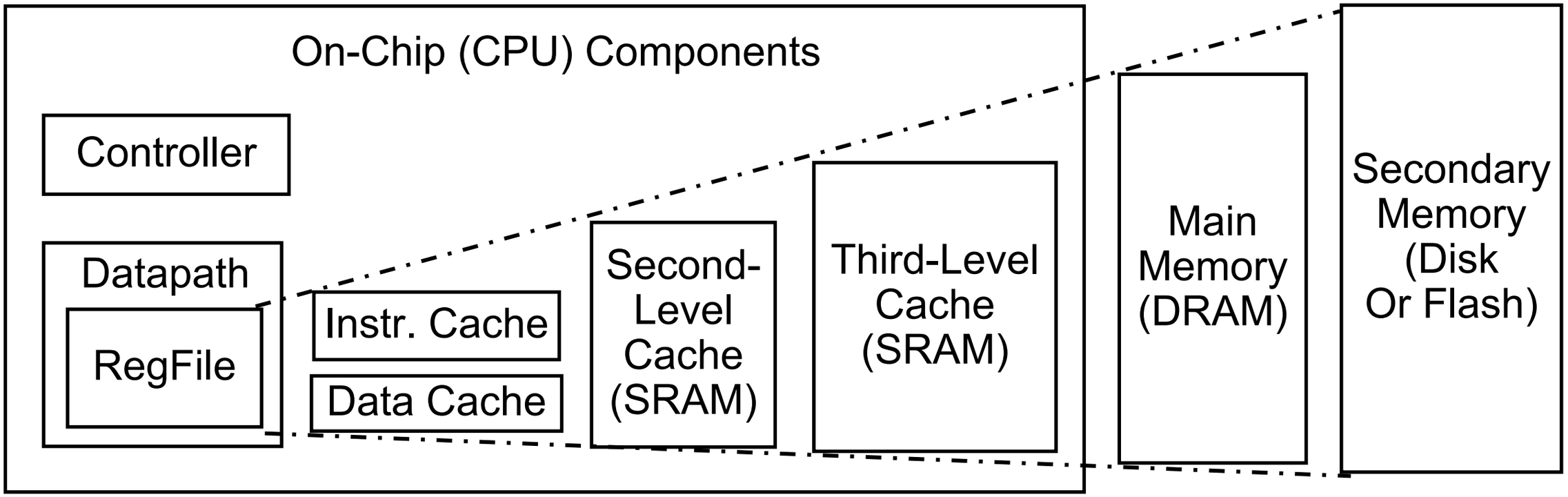
- Hit rate: fraction of accesses that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a line/block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)
- **Average Memory Access Time (AMAT)** is the average time to access memory considering both hits and misses in the cache

AMAT = Time for a hit + Miss rate × Miss penalty

AMAT

- **Average Memory Access Time (AMAT)** is the average time to access memory considering both hits and misses in the cache

AMAT = Time for a hit + Miss rate × Miss penalty

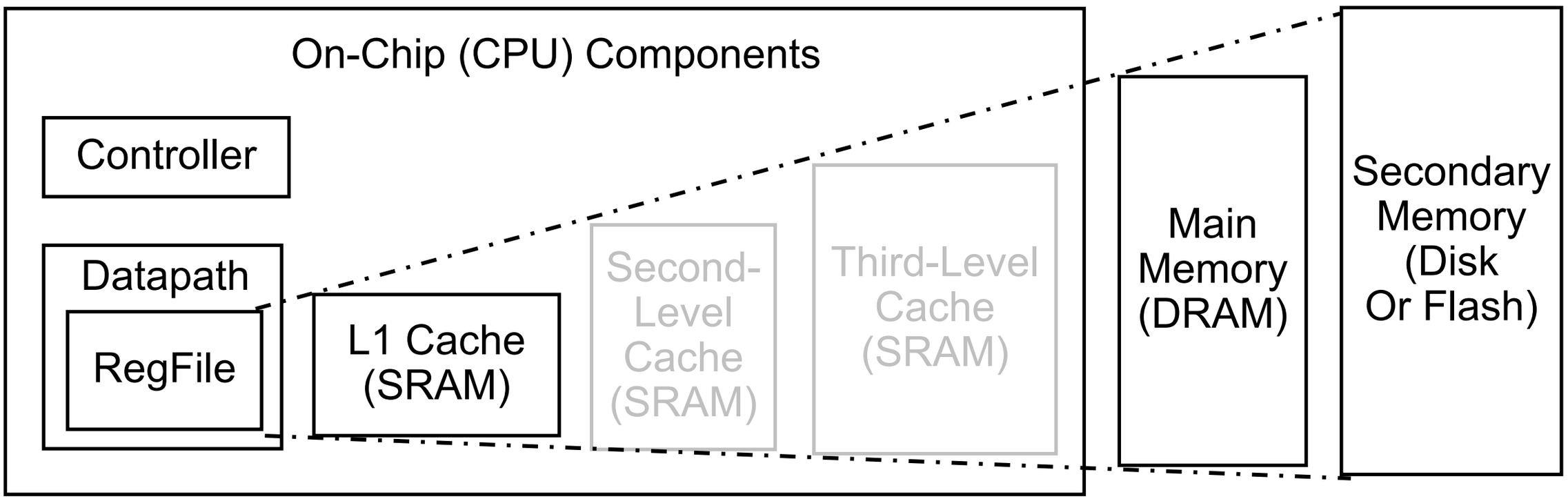


- Each level may have different hit time, miss rate and miss penalty
- To reduce miss rate and miss penalty

AMAT: Single-Level Cache

- Assume L1 cache only: hit time 1 cycle; miss rate 5%; miss penalty 100 cycles;

AMAT = Time for a hit + Miss rate × Miss penalty

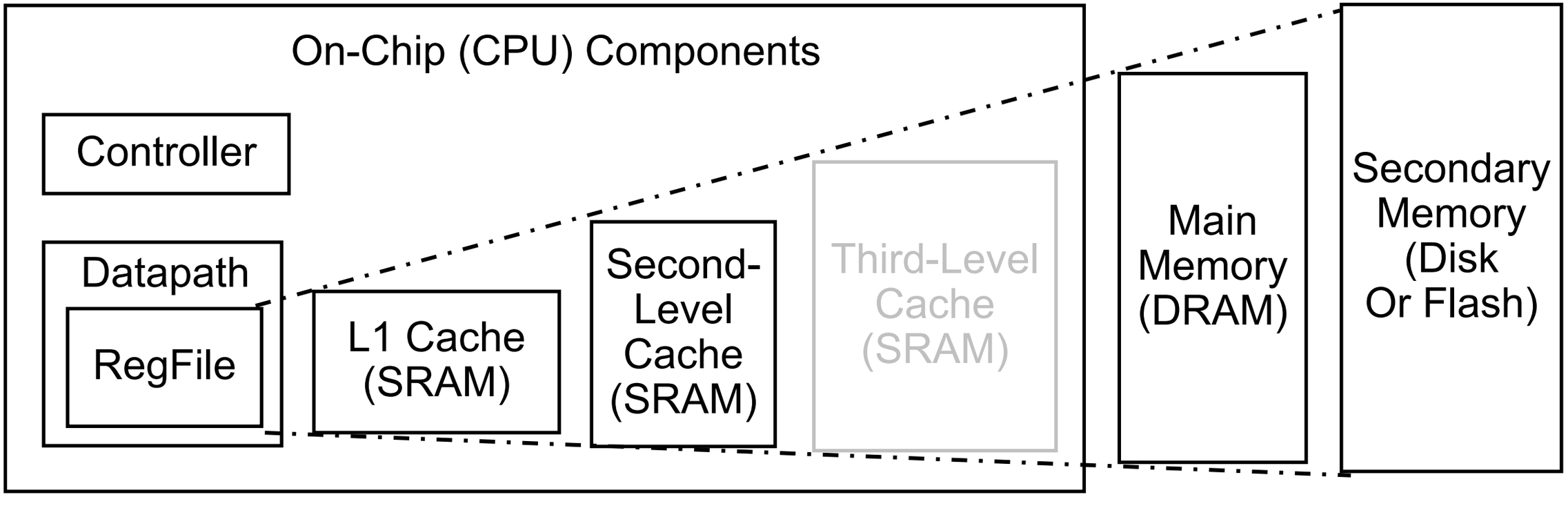


AMAT = 1 clock cycle + 5% × 100 clock cycles = 6 clock cycles

AMAT: Multi-Level Cache

- Assume L1 & L2 cache only:
- L1 hit time 1 cycle; miss rate 5%;
- L2 hit time 5 cycles, miss rate 10%, miss penalty 100 cycles;

L1 AMAT = Time for an L1 hit + L1 Miss rate × L2 AMAT



L2 AMAT = 5 clock cycles + 10% × 100 clock cycles = 15 clock cycles

AMAT = 1 clock cycle + 5% × 15 clock cycles = 1.75 clock cycles

AMAT Pitfall: Local vs. Global Miss Rate

- *Local miss rate* – the fraction of references to one level of a cache that miss

  Ø E.g. Local Miss rate L2$ = $L2 Misses / L1$ Misses

- *Global miss rate* – the fraction of references that miss in all levels of a multilevel cache
  - L2$ local miss rate >> than the global miss rate

E.g. Global Miss rate (two-level cache) = L2$ Misses/Total Accesses
= (L2$ Misses / L1$ Misses) × (L1$ Misses / Total Accesses)
= Local Miss rate L2$ × Local Miss rate L1$

AMAT = Time for a hit + Miss rate × Miss penalty
AMAT = Time for a L1 cache hit + (local) Miss rate L1 cache × (Time for a L2 cache hit + (local) Miss rate L2 cache × L2 cache miss penalty)

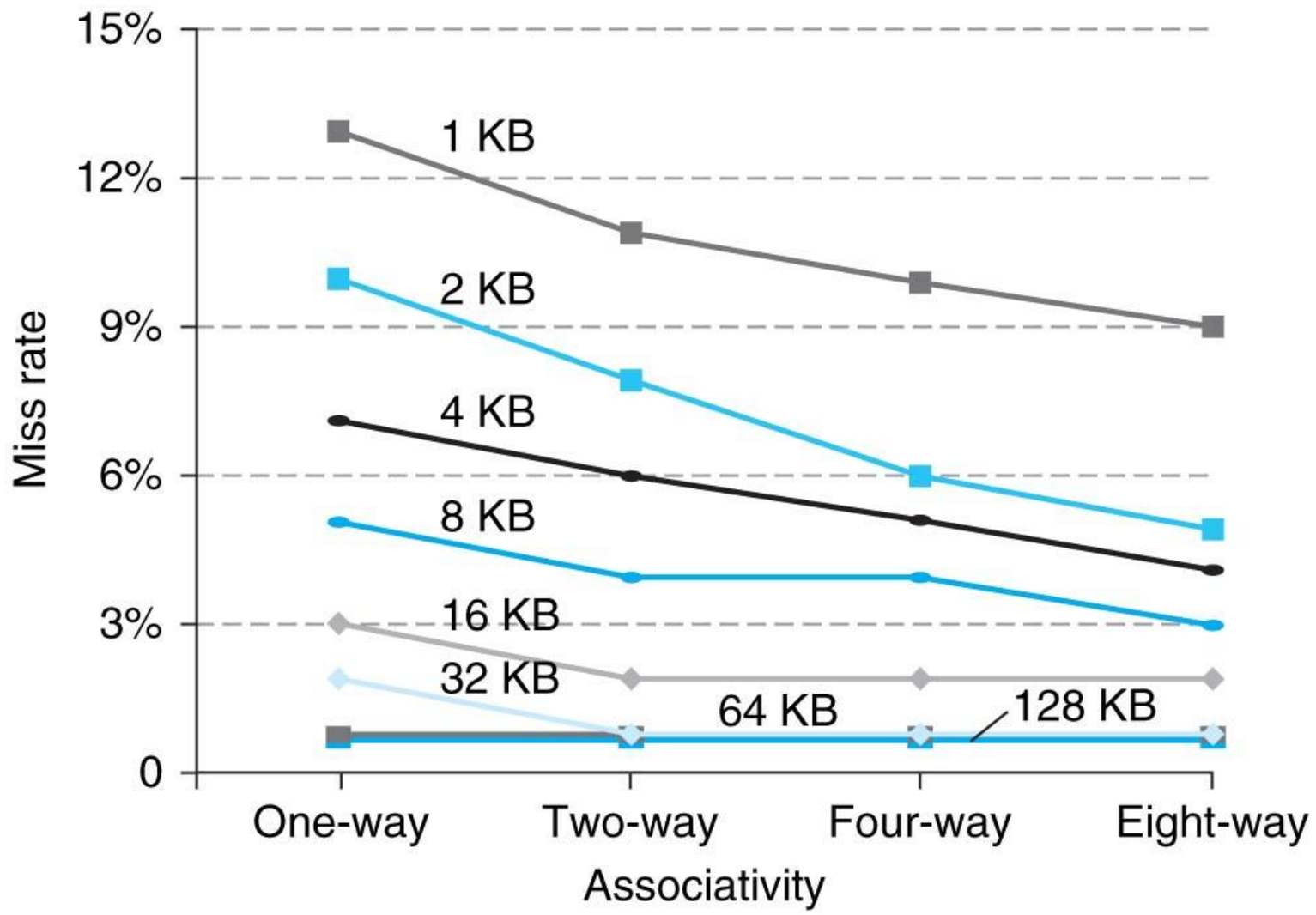
Improve Cache Performance

- Reduce time for a hit, miss rate and miss penalty (additional time cost compared to cache hit);
  - Reduce hit time: use smaller cache but may increase miss rate (capacity miss)
  - Reduce miss rate: this is complex
    - Program dependent;
    - Larger capacity (may decrease capacity miss but increase hit time and hardware cost);
    - Higher associativity (may reduce conflict miss; but require extra considertaion for replacement policy and increase hardware cost)
    - Larger cache blocks (may reduce compulsory miss; better spatial locality usage; but may harm temporal locality with recently-used data evicted)

Example of Cache Optimization

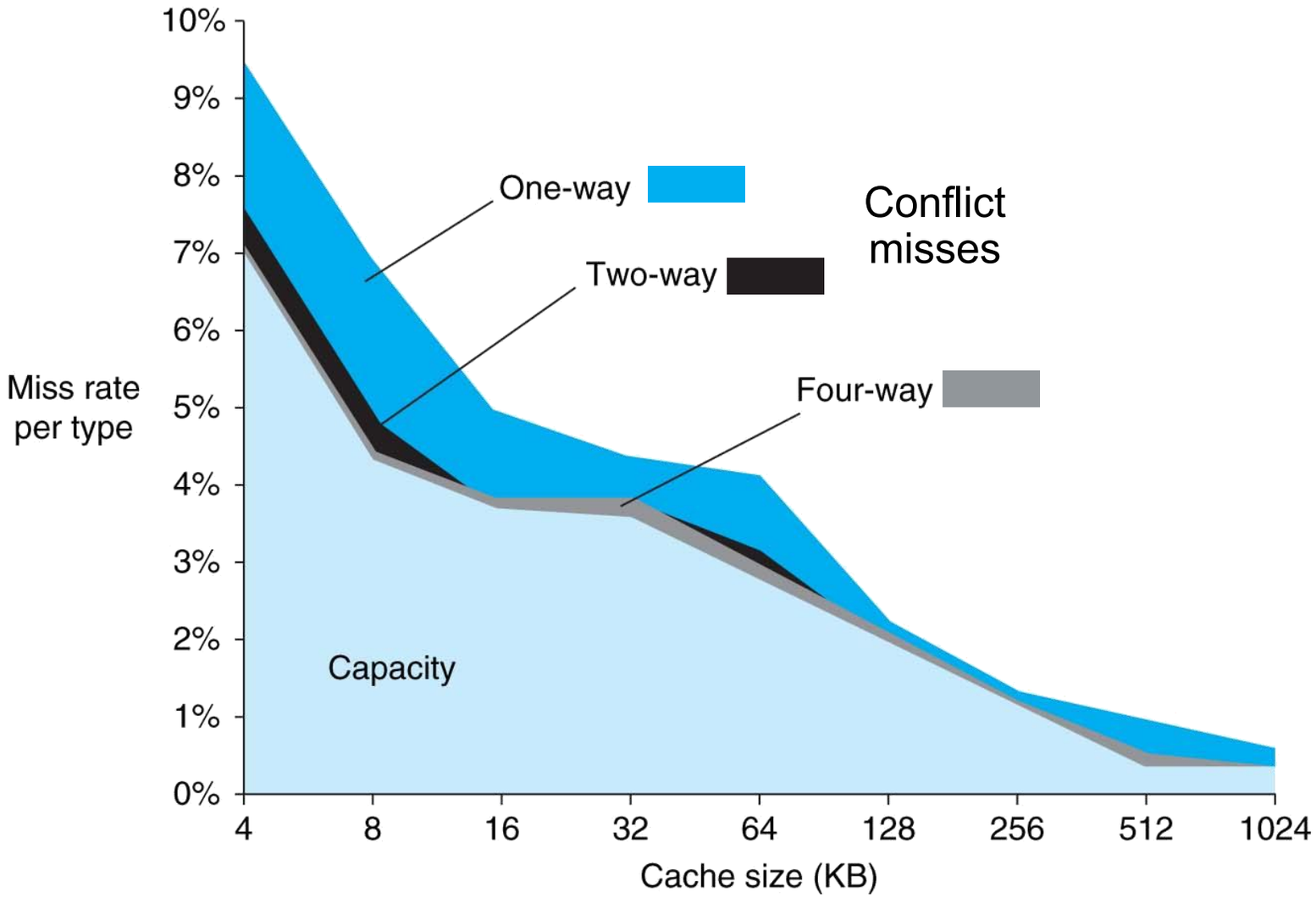
- Choice of DM cache versus SA cache depends on the cost of a miss versus the cost of implementation;

Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)



3Cs Analysis

- Three sources of misses (SPEC2000 integer and floating-point benchmarks)
  - Compulsory misses 0.006%, not visible
  - Capacity misses, function of cache size
  - Conflict portion depends on associativity and cache size

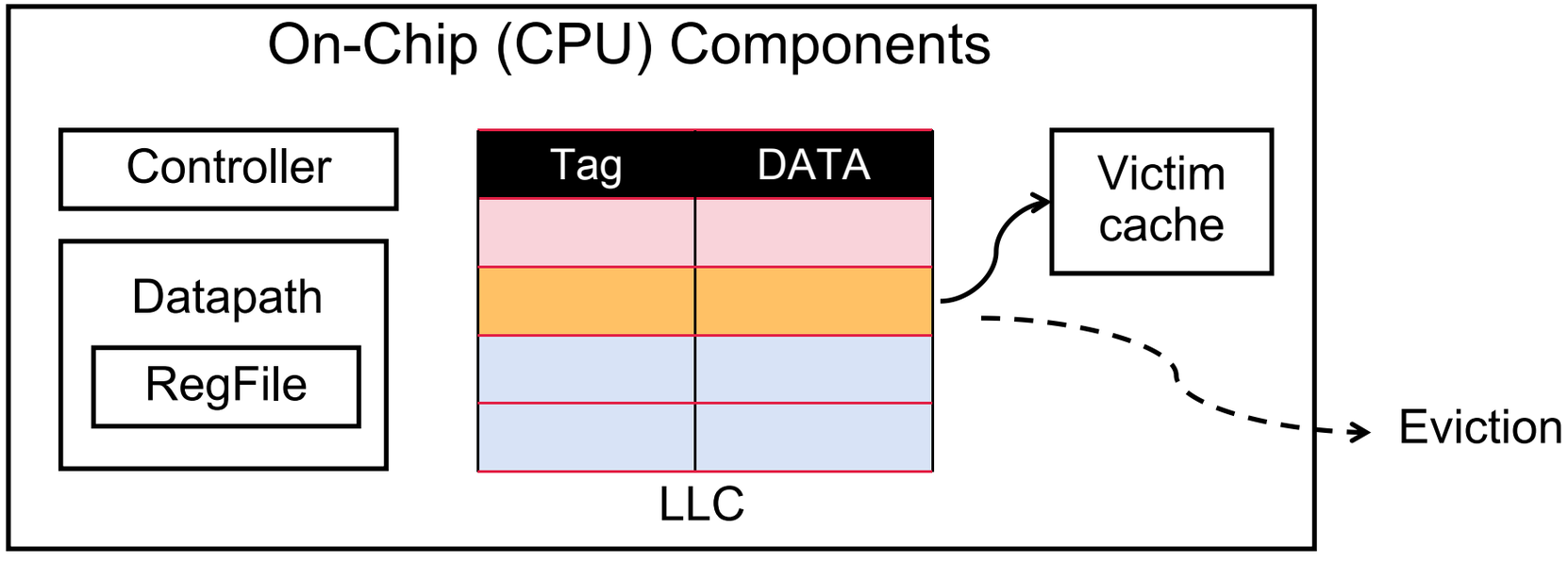


Improve Cache Performance

- Reduce time for a hit, miss rate and miss penalty (additional time cost compared to cache hit);
  - Reduce miss penalty:
    - Prefetch
      - programmer/compiler: I know that, later on, I will need this data;
      - Can be as an explicit prefetch instruction;
      - or an implicit instruction: lw x0 0(t0)
      - or hardware prefetcher;
      - Won't stall the pipeline on a cache miss: The processor control logic recognizes this situation;
      - Allows you to hide the cost of cache misses;
      - But favors big cache;

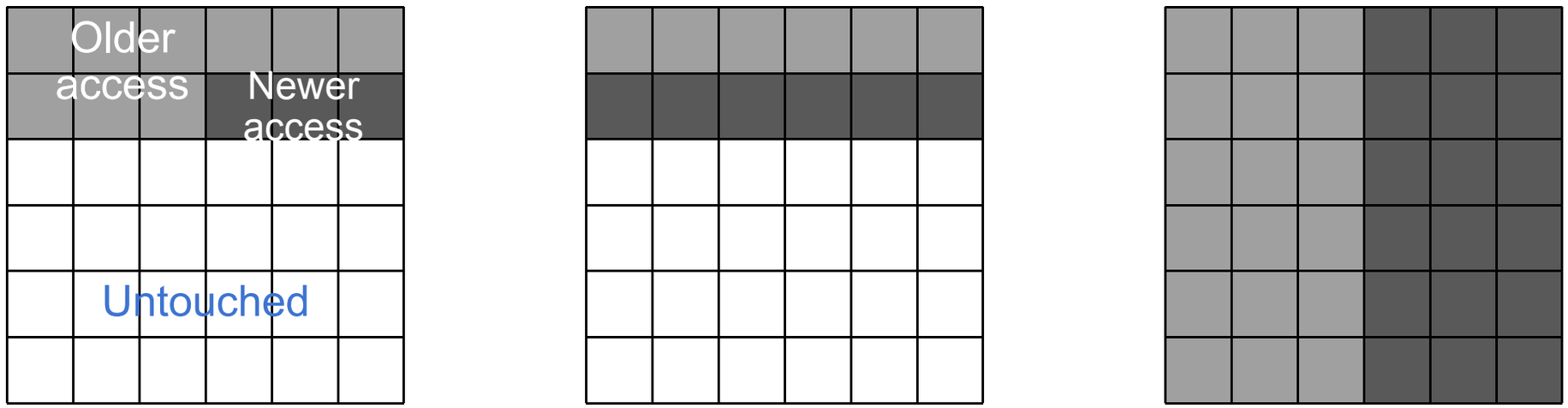
Improve Cache Performance

- Reduce time for a hit, miss rate and miss penalty (additional time cost compared to cache hit);
  - Reduce miss penalty:
    - Increase level of cache
      - Refer to previous example
      - But significantly increase hardware cost
    - Victim cache
      - Optionally have a very small (16-64 entry) ***fully associative*** "victim" cache



Cache-aware Program Optimization

- Cache blocking: GEMM example C = A x B (row-major)

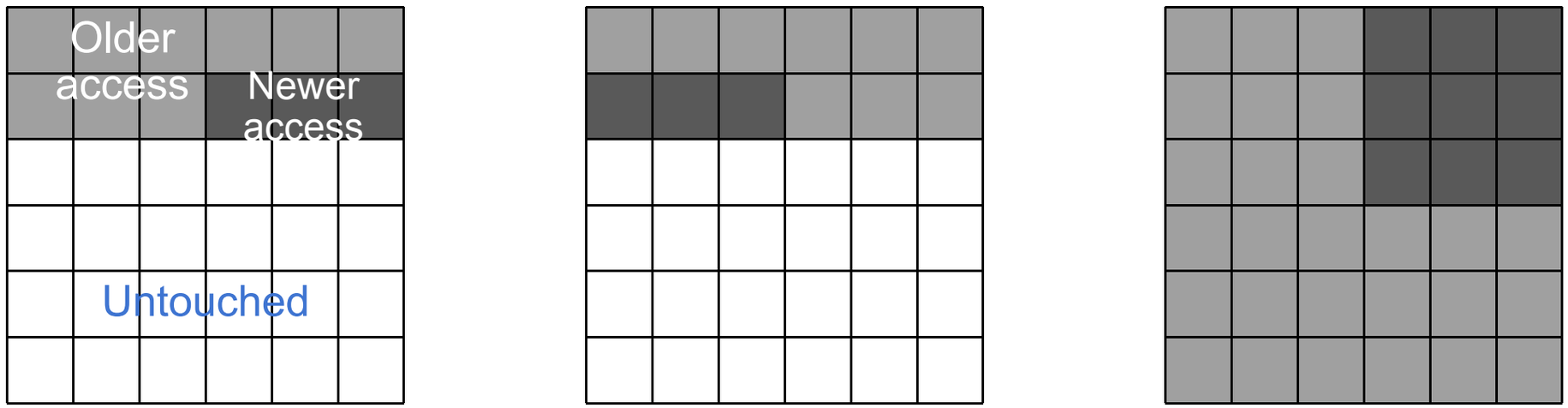



```
for (int i = 0; i<n; i++){
for (int j = 0; j<n; j++)
   {
      int cij = C[i*n+j];
      for (int k = 0; k < n; k++)
      {
         cij += A[i*n+k] * B[k*n+j];
         C[i*n+j] = cij;
      }
   }}
```

Assume a cache with a size of 8 ints., might incur a lot of misses

Cache-aware Program Optimization

- Cache blocking: GEMM example C = A x B (row-major)

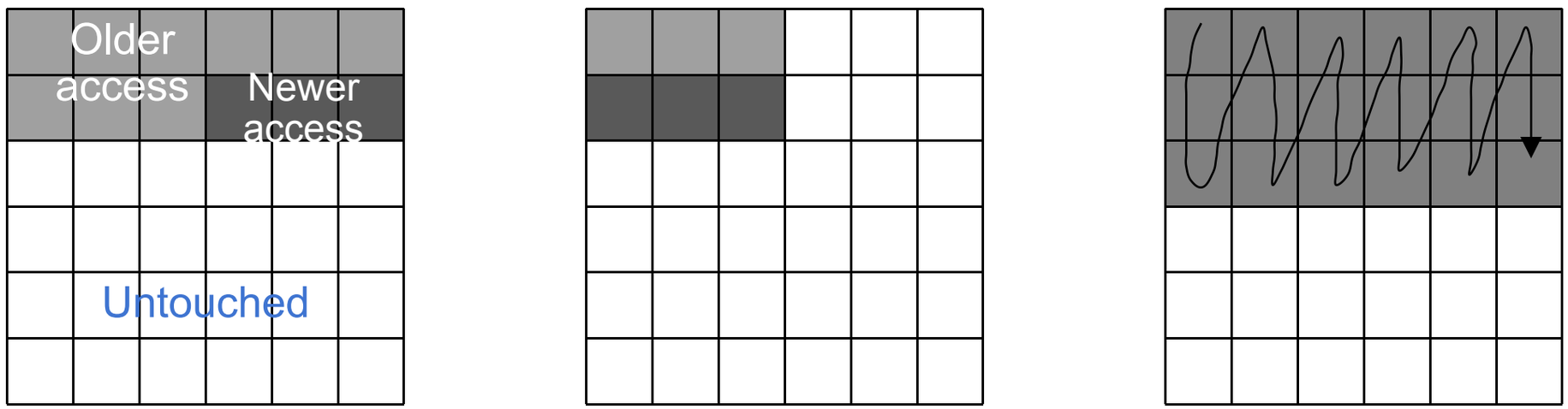



```
for (int i = 0; i<n; i++){
for (int j = 0; j<n; j++)
   {
      int cij = C[i*n+j];
      for (int k = 0; k < n; k++)
      {
         cij += A[i*n+k] * B[k*n+j];
         C[i*n+j] = cij;
      }
   }}
```

```
for (int i = 0; i < n; i++){
for (int j = 0; j < n; j++)
   {
      int cij = C[i*n+j];
      for (int blk_k = 0; blk_k < blk_num; blk_k++)
      {  for (int sk=0; sk < blk_size; sk++){
         cij += A[i*n+blk_k*blk_size+sk] *
                B[(blk_k*blk_size+sk)*n+j];
         C[i*n+j] = cij;}
      }
   }}   %blk_num = 2; sk: 0-2
```

Cache-aware Program Optimization

- Cache blocking: GEMM example C = A x B (row-major)




```
for (int blk_k = 0; blk_k < blk_num; blk_k++){
for (int i = 0; i < n; i++){
for (int j = 0; j < n; j++)
   {
      int cij = C[i*n+j];
      for (int blk_k = 0; blk_k < blk_num; blk_k++)
      {   for (int sk=0; sk < blk_size; sk++){
          cij += A[i*n+blk_k*blk_size+sk] *
                 B[(blk_k*blk_size+sk)*n+j];
          C[i*n+j] = cij;}
      }
}}}  blk_num = 2; sk 0-2;   blk_k: 0-1
```

Better A reuse and less A cache miss

Real Stuff

Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 cache associativity	4-way (I), 8-way (D) set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 cache associativity	16-way set associative	32-way set associative
L3 replacement	Not Available	Evict block shared by fewest cores
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

Real Stuff-GPU

- L0/L1 instruction cache;
- L1 data cache, private to each streaming multi-processor (SM) mixed with configurable shared memory (scratchpad memory, managed explicitly by the programmers);
- L2 unified cache for all SMs;



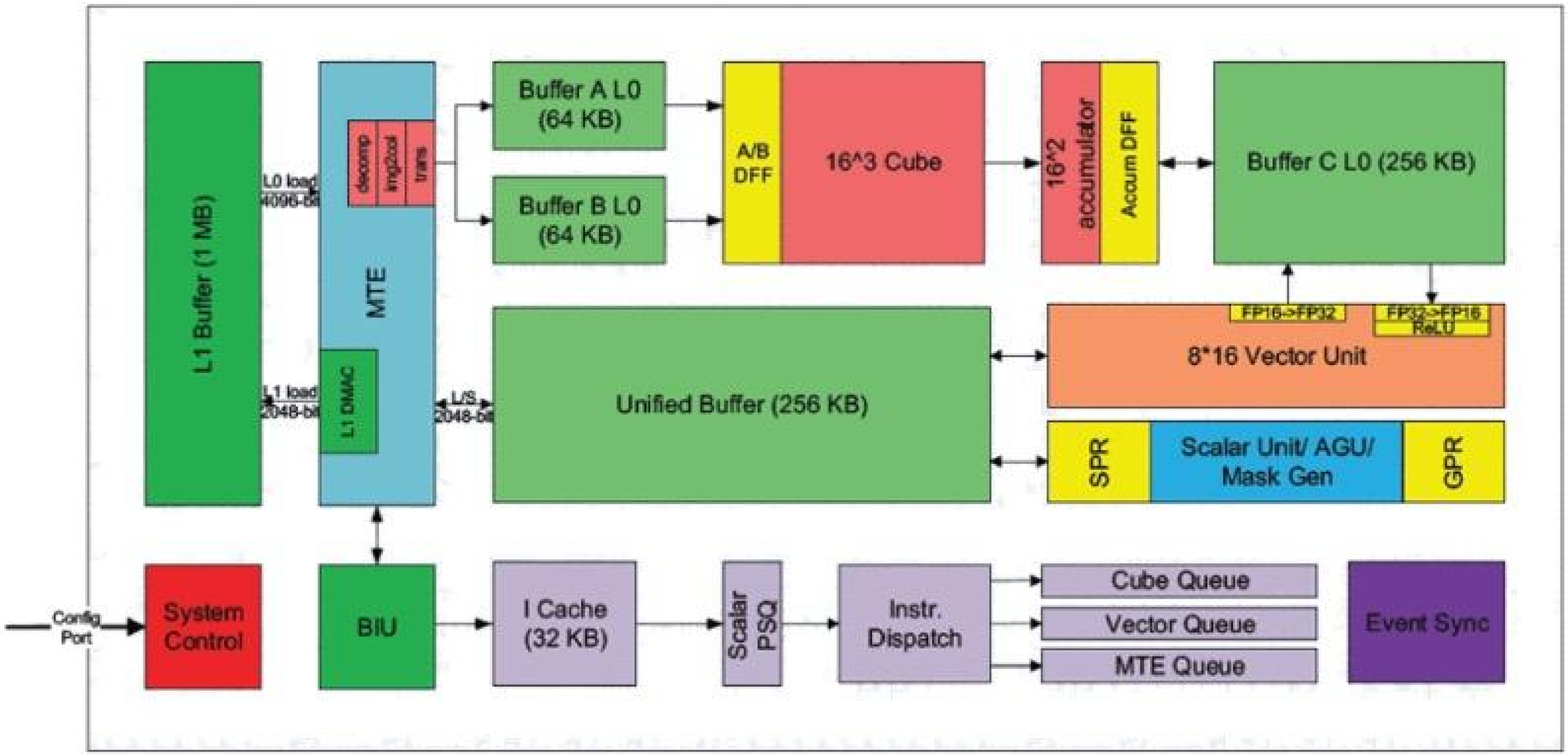
More in EE219 AI computing systems

One streaming multi-processor inside an H100 GPU



Credit to Nvidia

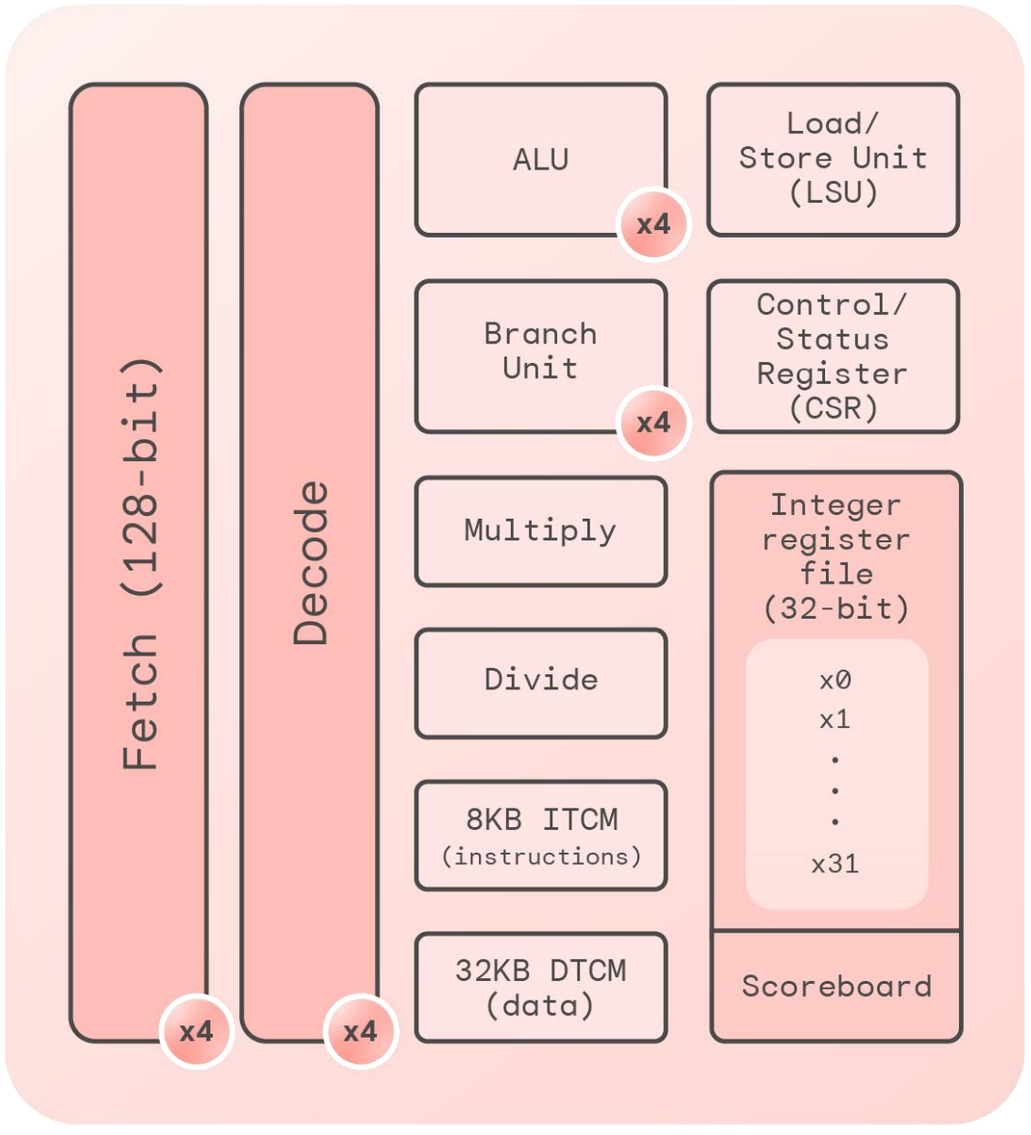
Real Stuff-NPU



More in EE219 AI computing systems

Credit to Huawei

Real Stuff-NPU

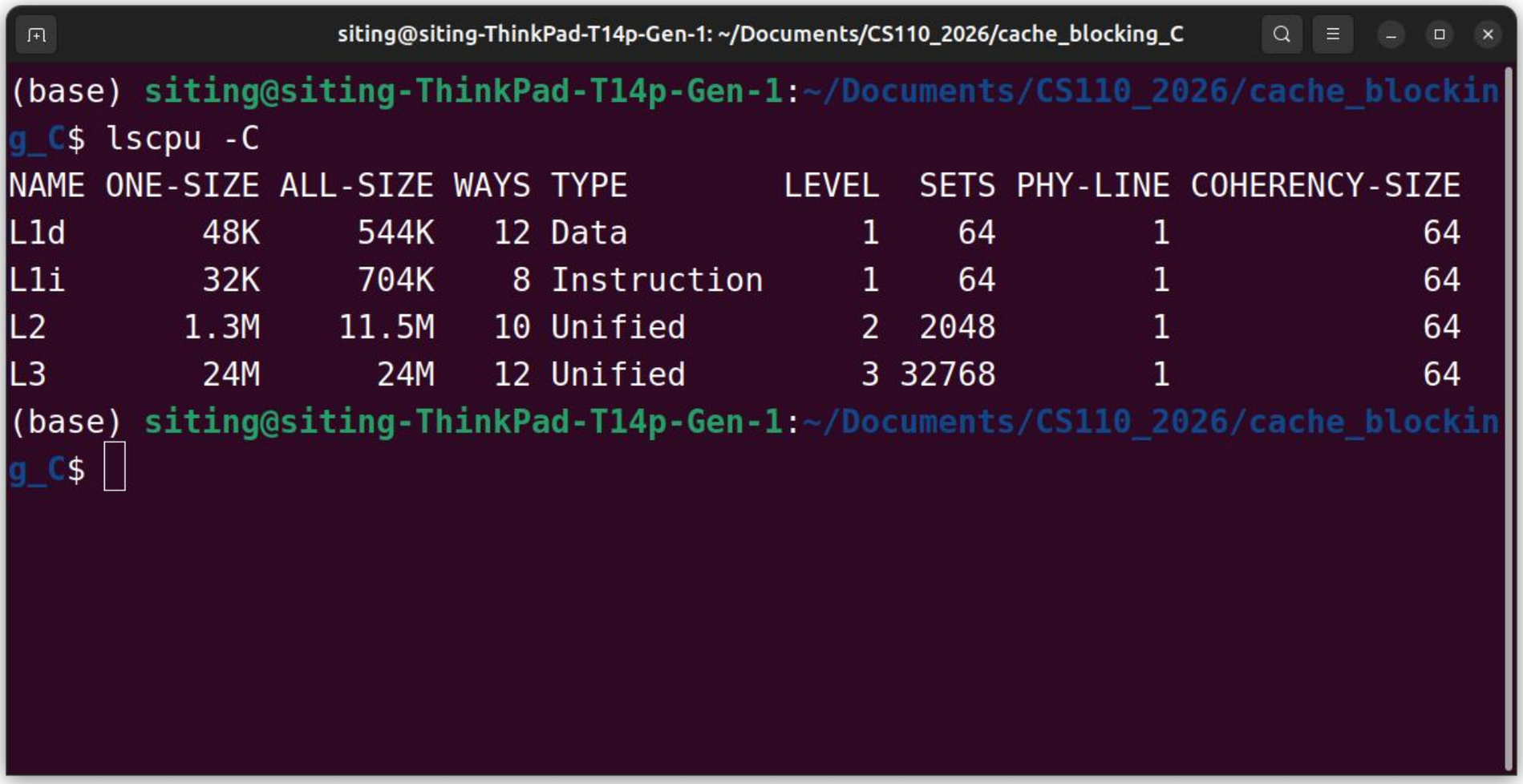


Scalar Core frontend
(RISC-V)

Credit to Google, CoralNPU

On your Computer

- On the start-up of Linux kernel, it probes the hardware and records hardware info (dynamically) in /sys/devices/system/...
- We can gain insight of the CPU & cache through this by either
- *lscpu -C*



On your Computer

- On the start-up of Linux kernel, it probes the hardware and records hardware info (dynamically) in /sys/devices/system/...
- We can gain insight of the CPU & cache through this by or directly
- cat /sys/devices/system/cpu/cpu0/cache/index0/...



Summary

- Fully associative cache: cache block can go anywhere, no index field, but requires 1 comparator/block;
- Direct mapped cache: cache block can go exactly one block, requires only 1 comparator; #sets = #blocks; no replacement policy required;
- N-way set-associative cache: N places for a cache block; #sets = #blocks/N; requires N comparators;
- Replacement policy: LRU; write policy: write-back vs. write-through, (non-)write-allocate on write miss;
- Cache performance and optimization: reduce the miss rate and penalty mainly